Intel® Edison Kit for Arduino*

Hardware Guide

December 2014

Revision 004

Document Number: 331191-004

Notice: This document contains information on products in the design phase of development. The information here is subject to change without notice. Do not finalize a design with this information.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication, or disclosure is subject to restrictions stated in Intel's Software License Agreement, or in the case of software delivered to the government, in accordance with the software license agreement as defined in FAR 52.227-7013.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

The code names presented in this document are only for use by Intel to identify products, technologies, or services in development that have not been made commercially available to the public, i.e., announced, launched, or shipped. They are not "commercial" names for products or services and are not intended to function as trademarks.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at *http://www.intel.com/design/literature.htm*.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See *http://www.intel.com/products/processor_number* for details.

Intel, the Intel logo, and Intel Atom are trademarks of Intel Corporation in the United States and other countries.

* Other brands and names may be claimed as the property of others.

Copyright © 2014 Intel Corporation. All rights reserved.

Contents

1 Introduction 6
- 1.1 Software requirements 6
- 1.2 Terminology 6
- 1.3 References 6

2 Product Overview 7
- 2.1 Shield pin GPIO mapping 8
- 2.2 Pin function multiplexing control (summary) 9
- 2.3 Pin function multiplexing control (detailed) 10
- 2.4 GPIO interrupt support 11
- 2.5 Miscellaneous GPIOs 11
- 2.6 Pin direction and pullup control 12

3 High-Level Functional Description 13
- 3.1 Intel® Edison kit for Arduino* header signal list 14
- 3.2 Intel® Edison kit for Arduino* PWM swizzler 15
- 3.3 Intel® Edison kit for Arduino* analog inputs 16
- 3.4 Intel® Edison kit for Arduino* signal pullup resistors 16
- 3.5 Intel® Edison kit for Arduino* USB interface 16
- 3.6 Intel® Edison kit for Arduino* power supply 17
- 3.7 Intel® Edison kit for Arduino* expansion mechanicals 17

4 Powering the Intel® Edison kit for Arduino* 18
- 4.1 Boot voltage selection – DCIN signal 19

5 Batteries 20

6 Layout 21
- 6.1 Antenna keepout 21
- 6.2 Layout SD card, I2S, SPI, I2C 21
- 6.3 LEDs 22

7 Handling 23

8 Debug UART and Low-Power Sleep Mode 24

9 Buttons 25
- 9.1 Software recovery (FWR_RCVR and RCVR_MODE) 25

10 Digikey sources 26

11 Shield pin configuration 27
- 11.1 Configure IO5 as a GPIO input, with pullup resistor disabled 27
- 11.2 Configure IO11 as a GPIO input, with pullup resistor disabled 28
- 11.3 Configure IO7 as a GPIO input, with pullup resistor enabled 28
- 11.4 Configure IO6 as a PWM output 29
- 11.5 Configure IO14 as an ADC input 29
- 11.6 Configure IO18/IO19 for I^2C connectivity 30
- 11.7 Configure IO10 through IO13 for SPI connectivity 31

Figures

Figure 1 Arduino* Uno* pinout 7
Figure 2 Intel® Edison kit for Arduino* block diagram 13
Figure 3 Intel® Edison kit for Arduino* PWM swizzler 15
Figure 4 PWM swizzler on the Intel® Edison kit for Arduino* 16
Figure 5 Intel® Edison kit for Arduino* mechanical dimensions 17
Figure 6 Intel® Edison kit for Arduino* power distribution network 18
Figure 7 Area around antenna 21
Figure 8 Inserting an Intel® Edison compute module 23
Figure 9 Digikey sources 26

Tables

Table 1 Product-specific documents 6
Table 2 Shield pin GPIO mapping 8
Table 3 Arduino* pin mux and pin mode settings 9
Table 4 Pin function multiplexing control 10
Table 5 GPIO interrupt support 11
Table 6 Miscellaneous GPIOs 11
Table 7 Pin direction and pullup control 12
Table 8 Intel® Edison kit for Arduino* header signal list 14
Table 9 Intel® Edison kit for Arduino* PWM swizzler signal assignments 15
Table 10 Layout SD card 21

Revision History

Revision	Description	Date
ww32	Initial release	August 4, 2014
ww34	Minor edits.	August 20, 2014
ww36	Removed a column from Table 2.	September 5, 2014
001	First public release.	September 9, 2014
002	Minor corrections.	September 15, 2014
003	Added product overview chapter and shield pin configuration chapter.	September 18, 2014
004	Updated sections on software recovery mode, pin function mux controls, and LEDs.	December 1, 2014

§

1 Introduction

This document describes the hardware interface of the Intel® Edison kit for Arduino*.

The kit contains external input/output pin connections which may be configured to be used in a variety of interfacing modes, such as GPIO, PWM, SPI, I²C, ADC, for compatibility with Arduino* Uno* shield hardware. This document describes the pin functions available, detailed GPIO pin mapping for pin control and I/O, and use of Linux command line tools to configure the external I/O pin functions correctly for the desired mode of operation.

1.1 Software requirements

- Intel® Edison kernel and BSP.
- Access to the Linux command line on an Intel® Edison compute module.

1.2 Terminology

Term	Definition
PWM	Pulse width modulation
GPIO	General purpose input/output
ADC	Analog to digital converter
SPI	Serial peripheral interface
I²C	Inter-integrated circuit

1.3 References

Table 1 Product-specific documents

Reference	Name	Number/location
331188	Intel® Edison Board Support Package User Guide	
331189	Intel® Edison Compute Module Hardware Guide	
331190	Intel® Edison Breakout Board Hardware Guide	
331191	Intel® Edison Kit for Arduino* Hardware Guide	(This document)
329686	Intel® Galileo and Intel® Edison Release Notes	
[GSG]	Intel® Edison Getting Started Guide	W: *https://communities.intel.com/docs/DOC-23147* M: *https://communities.intel.com/docs/DOC-23148* L: *https://communities.intel.com/docs/DOC-23149*
331438	Intel® Edison Wi-Fi Guide	

§

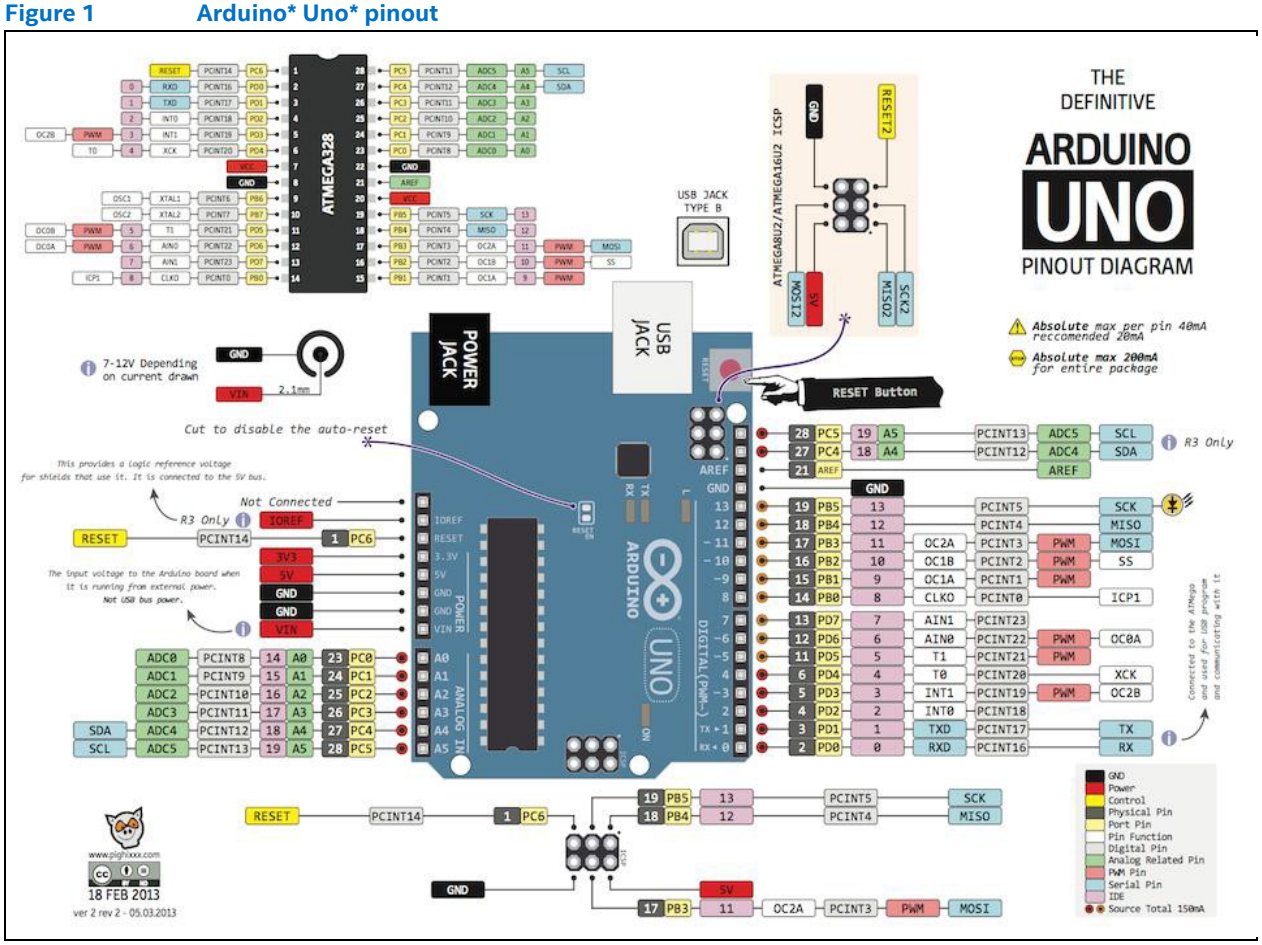
2 Product Overview

The 20 Arduino*-compatible shield I/O pins on the Intel® Edison kit for Arduino* are numbered IO0-IO19 (Figure 1). All pins support basic GPIO functionality. Some of the pins also support PWM, ADC, SPI or I^2C functions. Selection of different pin functions on the Intel® Edison kit for Arduino* is achieved through use of SoC pin control interfaces and GPIO output signals dedicated for multiplexing control. The following sections detail the mapping of each of the GPIO pins available on the Intel® Edison compute module to their respective functions, which can be broadly categorized as follows (see Figure 1):

- **External GPIO.** Used for digital input/output signaling via the external shield pins.
- **Pin multiplexing control.** Used for selecting different functions available on a given shield pin.
- **Pin buffer (level-shifter) direction control.** Used to configure the buffer on a given shield pin for input or output.
- **Pin pullup resistor control.** Used to enable/disable a pullup resistor on a given shield pin.

To use any of the supported functions on a shield pin, it is first necessary to configure the multiplexing, buffer direction, and pullup resistor controls applicable to that pin.

Figure 1 Arduino* Uno* pinout



2.1 Shield pin GPIO mapping

Table 2 describes the mapping of GPIO and PWM pin numbers (in Linux) to shield I/O pins. The following details are included:

- **Shield pin.** Digital I/O pin number as per Arduino* Uno* pin numbering scheme.
- **GPIO (Linux).** The pin number assigned under Linux.
- **Muxed functions.** Other signals available on this shield pin, as they appear on the schematic.

Table 2 Shield pin GPIO mapping

Shield pin	GPIO (Linux)	PWM (Linux)	Muxed functions	Notes
IO0	130		UART1_RXD	
IO1	131		UART1_TXD	
IO2	128		UART1_CTS	Note 1.
IO3	12	0	PWM0	Note 2.
IO4	129		UART1_RTS	Note 1.
IO5	13	1	PWM1	Note 2.
IO6	182	2	PWM2	Note 2.
IO7	48		—	
IO8	49		—	
IO9	183	3	PWM3	Note 2.
IO10	41	??	SPI_2_SS1	
			I2S_2_FS	Note 1.
			PWM4_OUT	Note 2.
IO11	43	??	SPI_2_TXD	
			I2S_2_TXD	Note 1.
			PWM5_OUT	Note 2.
IO12	42		SPI_2_RXD	
			I2S_2_RXD	Note 1.
IO13	40		SPI_2_CLK	
			I2S_2_CLK	Note 1.
IO14	44		AIN0	
IO15	45		AIN1	
IO16	46		AIN2	
IO17	47		AIN3	
IO18	14		AIN4	
			I2C_6_SDA	
IO19	165		AIN5	
			I2C_6_SCL	

1 *Some additional functions are available on certain SoC pins, such as I2S and UART flow control, but they are not currently supported by the Arduino library. However, it may be possible to use these from Linux.*

2 *Depends on PWM swizzler. The SoC offers only four PWM pins. A jumper pin matrix labeled "PWM swizzler" on the baseboard allows these four pins to be connected to any subset of the six shield-header pins normally used for PWM. From the factory, IO3, IO5, IO6, and IO9 will be connected to the four available SoC PWM pins as described above. You can manually alter these to connect IO10 or IO11.*

2.2 Pin function multiplexing control (summary)

All GPIO pins on the Arduino* header require some internal GPIOs to be set up before the pin is usable. This is usually as simple as setting an output enable, pullup enable, and mode. However, some pins have extra functionality such as SPI, PWM, or I^2C, so these pins need extra multiplexing (muxing) in order to be usable.

Table 3 shows this such that a programmer can easily see all the muxing pins affected for a given Arduino* header pin. The color codes in the table show related boxes. For example, the blue boxes are meant to show the relationship between the pin mux pins and the pin modes. This table is a synopsis of the more detailed tables below, which contain extra information, such as schematic pin numbers. For most needs, this synopsized table should suffice.

Table 3 Arduino* pin mux and pin mode settings

	Linux GPIO pin	GPIO pin mux			SoC pin modes		Output enable (high = output)	Pullup enable
		Linux pin	0 (low)	1 (high)	0	1	Linux	Linux
IO0	130				GPIO	UART	248	216
IO0	130				GPIO	UART	248	216
IO1	131				GPIO	UART	249	217
IO2	128				GPIO	UART	250	218
IO3	12				GPIO	PWM	251	219
IO4	129				GPIO	UART	252	220
IO5	13				GPIO	PWM	253	221
IO6	182				GPIO	PWM	254	222
IO7	48				GPIO		255	223
IO8	49				GPIO		256	224
IO9	183				GPIO	PWM	257	225
IO10	41	263	PWM	see 240	GPIO	I2S or SPI	258	226
		240	GPIO or I2S	GPIO or SPI_FS				
IO11	43	262	PWM	see 241	GPIO	I2S or SPI	259	227
		241	GPIO or I2S	GPIO or SPI TXD				
IO12	42	242	GPIO or I2S	GPIO or SPI RXD	GPIO	I2S or SPI	260	228
IO13	40	243	GPIO or I2S	GPIO or SPI CLK	GPIO	I2S or SPI	261	229
IO14 (A0)	44	200	GPIO	A0	GPIO		232	208
IO15 (A1)	45	201	GPIO	A1	GPIO		233	209
IO16 (A2)	46	202	GPIO	A2	GPIO		234	210
IO17 (A3)	47	203	GPIO	A3	GPIO		235	211
IO18 (A4)	14	204	GPIO or I2C SDA	A4	GPIO	I2C-6	236	212
IO19 (A5)	165	205	GPIO or I2C SCL	A5	GPIO	I2C-6	237	213

Note: Before setting up any muxing, set pin 214 (TRI_STATE_ALL) to LOW, make all of your changes, then set pin 214 to HIGH.

2.3 Pin function multiplexing control (detailed)

Table 4 lists the GPIO outputs dedicated to pin multiplexing control. Different functions may be selected for specific shield I/O pins by setting these GPIO outputs to 0/1 (low/high). Additionally, some of the SoC GPIO pins also feature internal mux options. These are listed as "SoC Pin Modes".

Currently, these are configured by setting the required pin mode for the corresponding SoC GPIO pin N, via */sys/kernel/debug/gpio_debug/gpioN/current_pinmux*, to "mode[0/1/2/...]"

Table 4 Pin function multiplexing control

Shield pin	GPIO pin mux					SoC pin modes				
	Pin	Linux	0 (low)	1 (high)	Power-on default	Pin	Linux	0	1	2
IO0	-					GP130	130	GPIO	UART	
IO1	-					GP131	131	GPIO	UART	
IO2	-					GP128	128	GPIO	UART	
IO3	-					GP12	12	GPIO	PWM	
IO4	-					GP129	129	GPIO	UART	
IO5	-					GP13	13	GPIO	PWM	
IO6	-					GP182	182	GPIO	PWM	
IO7	-					GP48	48	GPIO		
IO8	-					GP49	49	GPIO		
IO9	-					GP183	183	GPIO	PWM	
IO10	U34_ IO1.7	263	**PWM4_OUT**	GP41	Pulled down input	GP41	41	GPIO	I2S	
				SSP5_FS_1						
	U16_ IO1.0	240	GP41	**SSP5_FS_1**	Pulled up input [1]	GP111	111	GPIO	SPI	
IO11	U34_ IO1.6	262	**PWM5_OUT**	GP43	Pulled down input	GP43	43	GPIO	I2S	
				SSP5_TXD		GP115	115	GPIO	SPI	
	U16_ IO1.1	241	GP43	**SSP5_TXD**	Pulled up input [1]					
IO12	U16_ IO1.2	242	GP42	**SSP5_RXD**	Pulled up input [1]	GP42	42	GPIO	I2S	
						GP114	114	GPIO	SPI	
IO13	U16_ IO1.3	243	GP40	**SSP5_CLK**	Pulled up input [1]	GP40	40	GPIO	I2S	
						GP109	109	GPIO	SPI	
IO14	U17_ IO0.0	200	GP44	**A0**	Pulled up input [1]	GP44	44	GPIO		
IO15	U17_ IO0.1	201	GP45	**A1**	Pulled up input [1]	GP45	45	GPIO		
IO16	U17_ IO0.2	202	GP46	**A2**	Pulled up input [1]	GP46	46	GPIO		
IO17	U17_ IO0.3	203	GP47	**A3**	Pulled up input [1]	GP47	47	GPIO		
IO18	U17_ IO0.4	204	GP14	**A4**	Pulled up input [1]	GP14	14	GPIO	I2C-6	I2C-8
			I2C6_SCL			GP28	28	GPIO		
IO19	U17_ IO0.5	205	GP165	**A5**	Pulled up input [1]	GP165	165	GPIO	I2C-6	I2C-8
			I2C6_SDA			GP27	27	GPIO		

1. *These pins are pulled up inputs at power-on. This effectively enables the mux switches (i.e. mux function 1 is selected).*

2.4 GPIO interrupt support

All GPIO inputs on the Intel® Edison platform are interrupt-capable, and all interrupt types are supported on all inputs. Table 5 lists the specific edge- and level-triggered interrupt types that are supported on each pin.

Table 5 GPIO interrupt support

Shield pin	GPIO	Edge-triggered			Level-triggered [1]	
	Linux	Rising	Falling	Both	Low	High
IO0	130	Y	Y	Y	Y	Y
IO1	131	Y	Y	Y	Y	Y
IO2	128	Y	Y	Y	Y	Y
IO3	12	Y	Y	Y	Y	Y
IO4	129	Y	Y	Y	Y	Y
IO5	13	Y	Y	Y	Y	Y
IO6	182	Y	Y	Y	Y	Y
IO7	48	Y	Y	Y	Y	Y
IO8	49	Y	Y	Y	Y	Y
IO9	183	Y	Y	Y	Y	Y
IO10	41	Y	Y	Y	Y	Y
IO11	43	Y	Y	Y	Y	Y
IO12	42	Y	Y	Y	Y	Y
IO13	40	Y	Y	Y	Y	Y
IO14	44	Y	Y	Y	Y	Y
IO15	45	Y	Y	Y	Y	Y
IO16	46	Y	Y	Y	Y	Y
IO17	47	Y	Y	Y	Y	Y
IO18	14	Y	Y	Y	Y	Y
IO19	165	Y	Y	Y	Y	Y

1. Level-triggered interrupts are not supported by the Arduino library, a limitation of the GPIO sysfs interface.*

2.5 Miscellaneous GPIOs

The GPIOs listed in Table 6 are used for other platform functions and for Arduino shield compatibility.

Table 6 Miscellaneous GPIOs

Function	GPIO pin	GPIO Linux	Direction	Power-on default [1]	Initial setup
TRI_STATE_ALL	U17_IO1.6	214	Output	Pulled up input*	
SHLD_RESET	U17_IO1.7	215	Output	Pulled up input*	
SHLD_RESET	U17_IO0.7	207	Input	Pulled up input*	

1 These pins are pulled up inputs at power-on. In this state, they have the same effect as outputs set high.

2.6 Pin direction and pullup control

For most shield pins on the Intel® Edison kit for Arduino*, there is a buffer/level-shifter which needs to be configured for input or output direction, and an external 47 kohm pullup/pulldown resistor, which may be optionally enabled. Both are driven by dedicated GPIO outputs, listed in Table 7. When configuring a shield pin as an output, we advise configuring the buffer for output ***before*** setting the SoC GPIO pin direction to output. To disconnect the external pullup/pulldown resistors, it is necessary to configure as high-impedance inputs the GPIOs that drive them.

Note also that the GPIO signals from the PCAL9555A GPIO expanders have internal 100 kohm pullup resistors, which are connected to the GPIO pins by default. These need to be disabled in many cases, by configuring those pins as high-impedance inputs.

Table 7 Pin direction and pullup control

Shield pin	Output enable GPIO (high = output)			Pullup enable GPIO		
	Pin	Linux	Power-on default [1]	Pin	Linux	Power-on default [2]
IO0	U34_ IO0.0	248	Pulled-down input	U39_IO0.0	216	Pulled up input
IO1	U34_ IO0.1	249	Pulled-down input	U39_IO0.0	217	Pulled up input
IO2	U34_ IO0.2	250	Pulled-down input	U39_IO0.0	218	Pulled up input
IO3	U34_ IO0.3	251	Pulled-down input	U39_IO0.0	219	Pulled up input
IO4	U34_ IO0.4	252	Pulled-down input	U39_IO0.0	220	Pulled up input
IO5	U34_ IO0.5	253	Pulled-down input	U39_IO0.0	221	Pulled up input
IO6	U34_ IO0.6	254	Pulled-down input	U39_IO0.0	222	Pulled up input
IO7	U34_ IO0.7	255	Pulled-down input	U39_IO0.7	223	Pulled up input
IO8	U34_ IO1.0	256	Pulled-down input	U39_IO0.7	224	Pulled up input
IO9	U34_ IO1.1	257	Pulled-down input	U39_IO0.7	225	Pulled up input
IO10	U34_ IO1.2	258	Pulled-down input	U39_IO0.7	226	Pulled up input
IO11	U34_ IO1.3	259	Pulled-down input	U39_IO0.7	227	Pulled up input
IO12	U34_ IO1.4	260	Pulled-down input	U39_IO0.7	228	Pulled up input
IO13	U34_ IO1.5	261	Pulled-down input	U39_IO0.7	229	Pulled up input
IO14	U16_ IO0.0	232	Pulled-down input	U17_ IO1.0	208	Pulled up input
IO15	U16_ IO0.1	233	Pulled-down input	U17_ IO1.1	209	Pulled up input
IO16	U16_ IO0.2	234	Pulled-down input	U17_ IO1.2	210	Pulled up input
IO17	U16_ IO0.3	235	Pulled-down input	U17_ IO1.3	211	Pulled up input
IO18	U16_ IO0.4	236	Pulled-down input	U17_ IO1.4	212	Pulled up input
IO19	U16_ IO0.5	237	Pulled-down input	U17_ IO1.5	213	Pulled up input

1 These pins are externally pulled down inputs at power-on. This effectively selects input direction for level shifters.

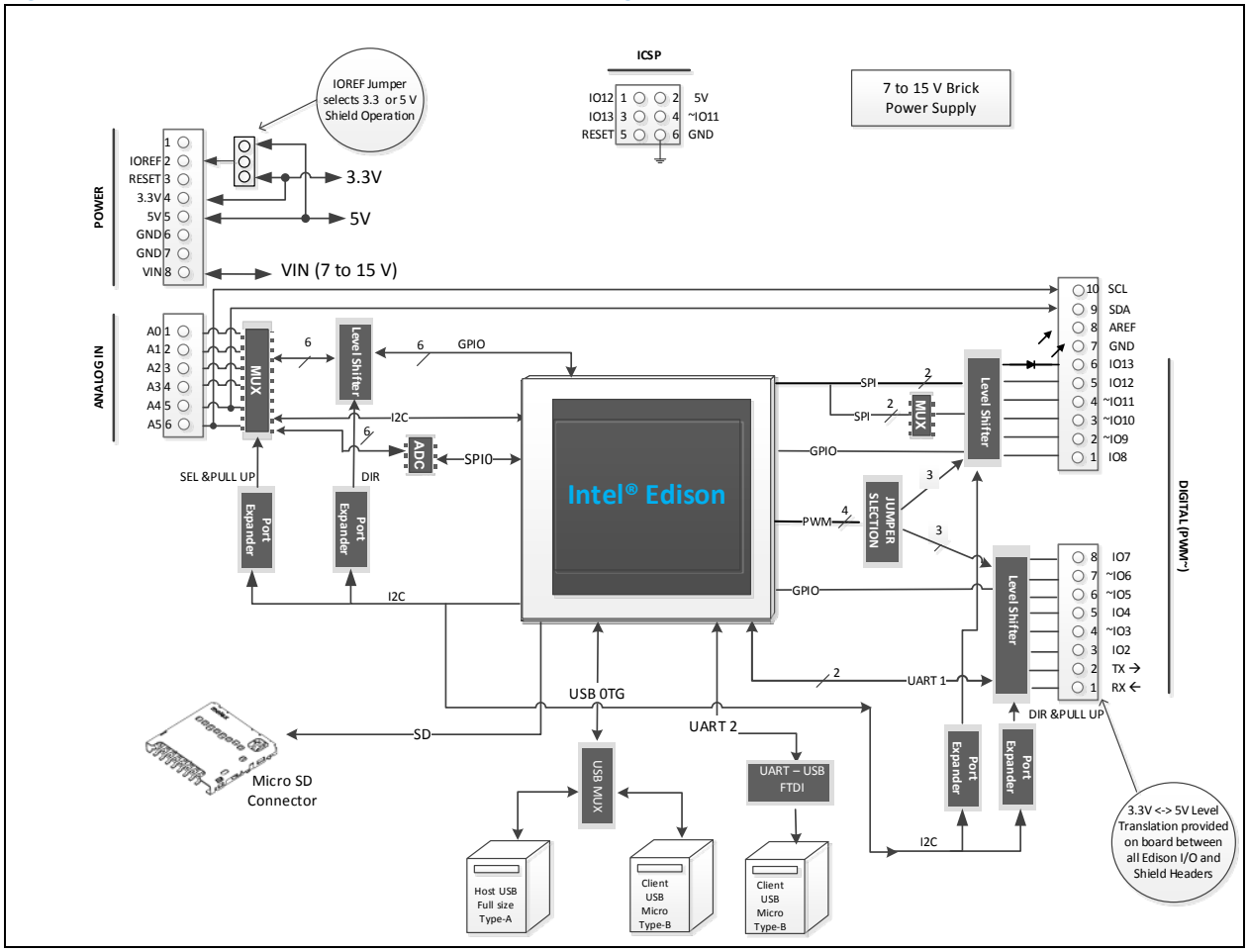
2 These pins are internally pulled up inputs at power-on. This effectively enables pullups (as 100 kohm + 47 kohm in series).

§

3 High-Level Functional Description

The Intel® Edison kit for Arduino*expansion board is designed to be hardware and software pin-compatible with Arduino shields designed for the Uno R3. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, ICSP header, and the UART port pins (0 and 1) are all in the same locations as on the Arduino Uno R3. This is also known as the Arduino 1.0 pinout. Additionally, the Intel® Edison kit for Arduino* board includes a micro SD card connector, a micro USB device port connected to UART2, and a combination micro USB device connector and dedicated standard size USB 2.0 host Type-A connector (selectable via a mechanical microswitch).

Figure 2 Intel® Edison kit for Arduino* block diagram



3.1 Intel® Edison kit for Arduino* header signal list

The Intel® Edison kit for Arduino* digital signals can be configured as input or output. When programmed as an input, a GPIO can serve as an interrupt. The Intel® Edison kit for Arduino* 1.8 V I/O are translated to 3.3 or 5 V using SN74LVC1T45 dual supply bus transceivers with 3 state outputs. Both outputs go tristate if either supply rail is at ground. The port direction is referenced to VCCA. The drive level for the transceiver is: ±4 mA at 1.8 V, ±24 mA at 3.3 V, and ±32 mA at 5 V.

Note: Drive level at 1.8 V is for reference only – pertains to drive level towards the Intel® Edison compute module.

Table 8 Intel® Edison kit for Arduino* header signal list

Header	Arduino pin name	Signal function
Power	N/C	Not connected
Power	IOREF	Shield I/O reference voltage (select 3.3 or 5 V via jumper on board)
Power	RESET	Shield reset (programmable via software or manual push button)
Power	3.3 V	System 3.3 V output
Power	5 V	System 5 V output
Power	GND	Ground
Power	GND	Ground
Power	VIN	System input power (7 to 15 V)
Analog	A0	Analog input or digital I/O
Analog	A1	Analog input or digital I/O
Analog	A2	Analog input or digital I/O
Analog	A3	Analog input or digital I/O
Analog	A4 / SDA	Analog input, digital I/O, or I^2C data (also connected to digital header)
Analog	A5 / SCL	Analog input, digital I/O, or I^2C data (also connected to digital header)
Digital	SCL	I^2C clock
Digital	SDA	I^2C data
Digital	AREF	ADC reference voltage (select AREF or IOREF via jumper J8 on board)
Digital	GND	Ground
Digital	13 / SCK	Digital I/O, or SPI clock
Digital	12 / MISO	Digital I/O, or SPI receive data
Digital	~11 / MOSI	Digital I/O, SPI send data, or PWM (configured with PWM swizzler)
Digital	~10	Digital I/O, SPI signal select, or PWM (configured with PWM swizzler)
Digital	~9	Digital I/O, PWM (configured with PWM swizzler)
Digital	8	Digital I/O
Digital	7	Digital I/O
Digital	~6	Digital I/O, PWM (configured with PWM swizzler)
Digital	~5	Digital I/O, PWM (configured with PWM swizzler)
Digital	4	Digital I/O
Digital	~3	Digital I/O, PWM (configured with PWM swizzler)
Digital	2	Digital I/O
Digital	1 / TX →	Digital I/O
Digital	0 / RX ←	Digital I/O
ICSP	MISO	SPI receive data (connected to digital pin 12)
ICSP	5V	System 5 V output
ICSP	SCK	SPI clock (connected to digital pin 13)
ICSP	MOSI	SPI send data (connected to digital pin 11)
ICSP	RESET	Shield reset (programmable via software or manual push button)
ICSP	GND	Ground

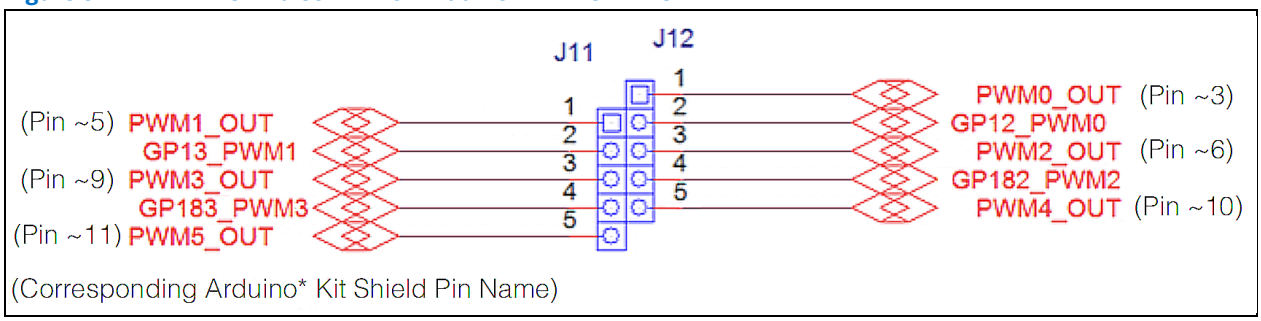
3.2 Intel® Edison kit for Arduino* PWM swizzler

There are four available GPIO that can be configured as PWM outputs. The PWM features are:

- The PWM Output Frequency and Duty Cycle can be estimated by the equations:
- Target frequency ~= 19.2 MHz * Base_unit value / 256
- Target PWM Duty Cycle ~= PWM_on_time_divisor / 256

The four PWM sources are wired to a PWM "swizzler". This pin header arrangement allows the four PWM sources to be routed to any four of the six Arduino header pins. Figure 3 shows the PWM swizzler.

Figure 3 Intel® Edison kit for Arduino* PWM swizzler



The four PWM sources from the Intel® Edison compute module (GP12_PWM0, GP13_PWM1, GP182_PWM2, and GP183_PWM3) can be configured to drive four of the six Arduino* header PWMs. Each Intel® Edison kit for Arduino PWM can be jumpered to one of three Arduino PWMs. For example, GP12_PWM0 can be jumpered to PWM0_OUT, PWM2_OUT, or PWM1_OUT.

Arduino* multiplexing has secondary multiplexing options of SPI (or GPIO). No other PWM has these secondary multiplexing options. Therefore, if the four Intel® Edison compute module PWMs are used and are not connected to the first four Arduino* PWM pins, then those unused pins of the first four pins cannot be used as a GPIO. They will have any function; they cannot be inputs or outputs (Table 9).

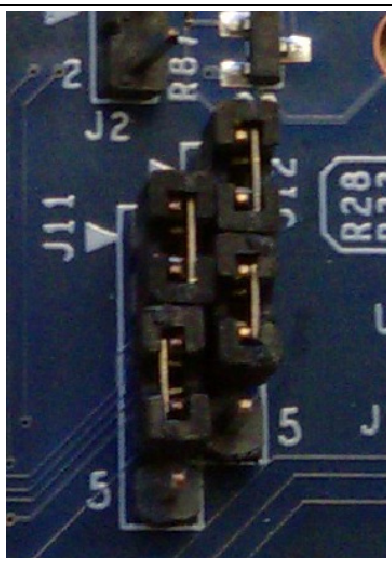
Table 9 Intel® Edison kit for Arduino* PWM swizzler signal assignments

Digital pin	Uno	Uno	Edison I/O	Edison PWM
11	IO	PWM(5)	GP43 (SSP2_TXD)	PWM3
10	IO	PWM(4)	GP41 (SSP2_FS0)	PWM3, PWM2
9	IO	PWM(3)	GP183_PWM3	PWM3, PWM2, PWM1
6	IO	PWM(2)	GP182_PWM2	PWM2, PWM1, PWM0
5	IO	PWM(1)	GP13_PWM1	PWM1, PWM0
3	IO	PWM(0)	GP12_PWM0	PWM0

The factory default jumper configuration of Intel® Edison kit for Arduino* has digital pins 3, 5, 6, and 9 attached to GPx_PWMx. These pins can be configured to be either a GPIO or a PWM output. The swizzler allows the four Intel® Edison compute module PWMs to be mapped to the six Arduino* pins as shown in the last column of Table 9. For example, if PWM0 is mapped to digital pin 5, then there is no Intel® Edison kit for Arduino* pin available to connect to Digital pin 3. So this pin no longer has a function. If it is driven as an output, it will output high. If it is driven as an input, the signal is lost in the swizzler.

The default configuration is DIG3 = GP12_PWM0, DIG5 = GP13_PWM1, DIG6 = GP182_PWM2, and DIG9 = GP183_PWM3. This requires jumpers on J12 1-2, and J12 3-4, J11 1-2, and J11 3-4, as shown in Figure 4.

Figure 4 PWM swizzler on the Intel® Edison kit for Arduino*



3.3 Intel® Edison kit for Arduino* analog inputs

The analog inputs are fed to an ADS7951 A/D converter. This device has the following features:

- 20 MHz clock rate
- 12-bit A/D conversion
- 1 MHz sample rate
- 70 dB signal to noise ratio
- 0 to 2.5 V or 0 to 5 V input range (select either AREF or IOREF via jumper J8 onboard)

The analog inputs are multiplexed with digital I/O using SN74LVC2G53 analog switches. These switches isolate the digital I/O from the analog input to prevent crosstalk. The SN74LVC2G53 also has an inhibit pin that places the I/O in a tristate condition. The switch also has low on state resistance of 15 ohm at 4.5 V VCC.

3.4 Intel® Edison kit for Arduino* signal pullup resistors

The analog and digital pins can be configured to have an external pull-up resistor connected. The pullup value is fixed at 47 kohm.

3.5 Intel® Edison kit for Arduino* USB interface

The Intel® Edison compute module has a single USB 2.0 interface. This interface is the primary method for downloading code. The Intel® Edison compute module is designed to support OTG, using the ID signal. Circuitry on the Intel® Edison kit for Arduino* board uses a USB multiplexer, and an external switch to configure the USB interface as a host port or device port. SW1 is a slider switch which selects between host mode and device mode. When the slider is switched towards the USB standard size Type A connector, the Intel® Edison compute module will go to host mode. When the switch is towards the micro USB Type B connector, the Intel® Edison compute module will go to device mode.

Note: USB host mode always requires use of an external power adapter.

3.6 Intel® Edison kit for Arduino* power supply

The Intel® Edison compute module is a low power device. In general it will not draw more than 200 mA (approximately 430 mA (final value TBD) when transmitting over Wi-Fi) from the main power source. Therefore, the Intel® Edison kit for Arduino* may run on USB power (when configured as a device), or off an external power adapter from 7 to 15 V.

Power from the external power adapter goes to a DC-DC converter and down converted to 5 V. The 5 V rail is diode-ORed with the USB micro B VBUS rail. This power goes to a DC-DC converter which down converts the power to 4.4 V. This voltage is in the safe range for the Intel® Edison compute module VSYS. The VSYS power range is 3.15 V min to 4.5 V max. This allows VSYS to run off a standard lithium ion battery.

The onboard charger IC is configured to detect the input power source and to limit the input power to either 500 mA (if connected to USB micro B port) or up to 1 A if connected to the DC power jack. The charger is programmed to charge at 100 mA. This charger is designed to charge standard lithium ion batteries with 4.2 V maximum charging voltage. End-users are responsible for choosing a suitable battery and following all safety precautions, to assure overcharging or charging when the battery temperature is too high is avoided.

For low power applications (those shields running off 3.3 V) a lithium ion battery (3.0 to 4.3 Vmax) can be attached to J2, which will power the Intel® Edison kit for Arduino and provide 100 mA of 3.3 V to the shield.

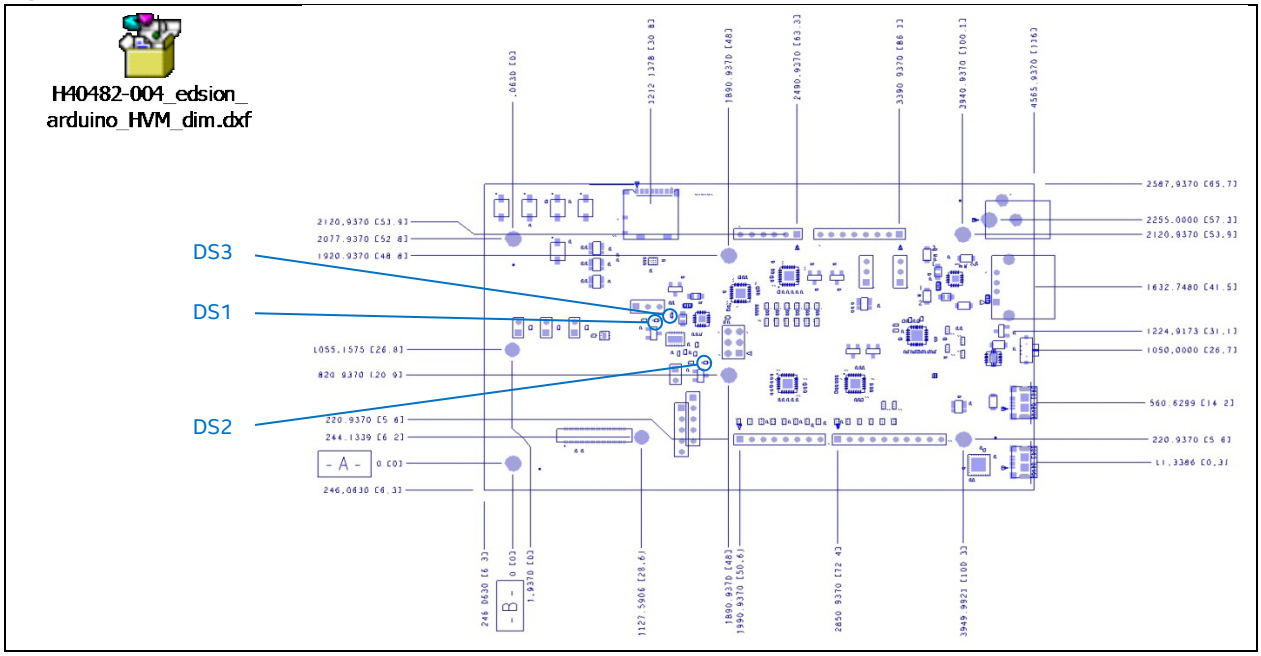
Some considerations of the power distribution in the Intel® Edison kit for Arduino*:

- Due to the diode ORing of the 5 V DC/DC and the VBUS input, means the 5 V power to the shield header will be nominally below 5 V. In the case of VBUS the voltage may be as low as 4.4 V (4.75 V VBUS min – 0.3 V diode drop. In the case of external power adapter 4.7 V.
- USB host mode always requires use of an external power adapter.

3.7 Intel® Edison kit for Arduino* expansion mechanicals

Figure 5 lists the dimensions (in thousands of inches and [mm]) of the Intel® Edison kit for Arduino* board.

Figure 5 Intel® Edison kit for Arduino* mechanical dimensions



§

4 Powering the Intel® Edison kit for Arduino*

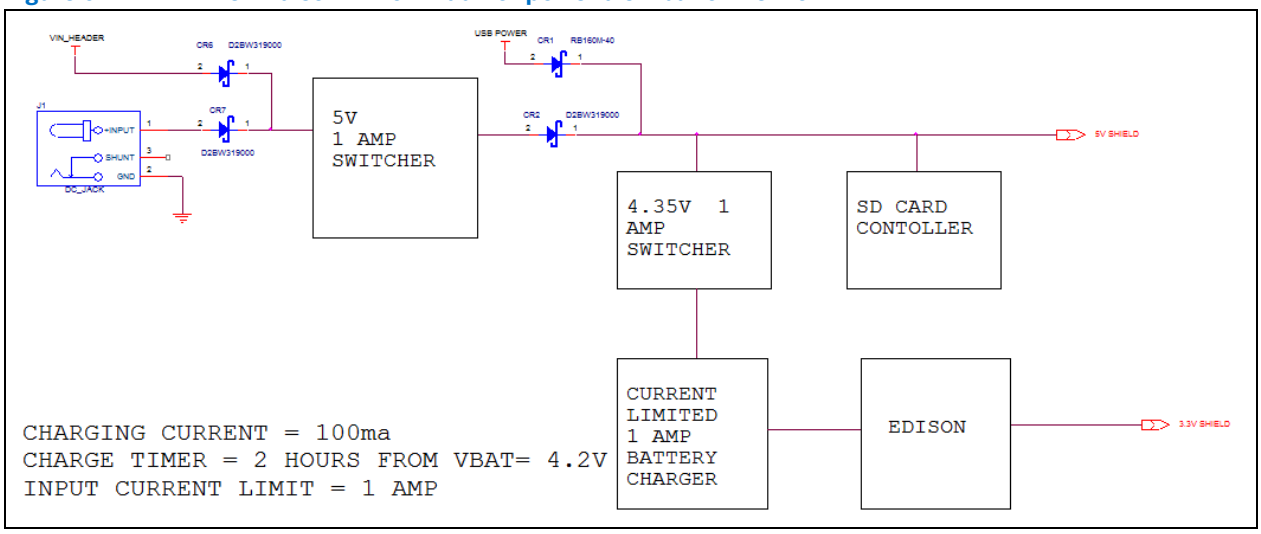
You can power the Intel® Edison kit for Arduino* using any of the following:

- an external power supply on J1;
- DCIN via shield header pin VIN;
- a USB cable via micro USB connector J16; or
- a lithium-ion battery connected to J2.

When power is applied to J1 or VIN, the external power must be in the range of 7 to 17 V. The power is converted to 5 V via a switching power supply, which powers the rest of the system. This supply was designed for a 1 A continuous supply. Higher currents will generate more power losses and may thermally damage the switcher. The switcher does have internal short circuit protection, and thermal shutdown protection. The end-user should not rely on thermal not short circuit protection.

Figure 6 shows the power distribution network of the Intel® Edison kit for Arduino*.

Figure 6 Intel® Edison kit for Arduino* power distribution network



Power from the 5 V switcher is diode-ORed with power from the USB connector. This arrangement allows the Intel® Edison kit for Arduino* to run off external power or USB power. This rail is used to power the shields, the SD card slot, and a 4.35 V switcher. The total current on this rail should be limited to 1 A maximum continuous.

The 4.35 V rail powers a battery charger and the Intel® Edison compute module. The 4.3 V supply is also designed to generate 1 A, and has the same protections (thermal and short circuit) as the 5 V supply.

The charger is designed to only accept 1 A maximum from the 4.35 V rail, and will charge a battery at 100 mA. The charger will supply power from the 4.35 V input or from the battery (if attached). The charger will charge the battery (from the 4.35 V supply) autonomously using whatever power is left over from powering the Intel® Edison kit for Arduino.

For low voltage systems, the Intel® Edison compute module can provide 3.3 V at 250 mA to the shields. The user should limit the current from the Intel® Edison kit for Arduino* 3.3 V rail. Higher currents will cause the 3.3 V output to droop (due to IR losses), and may cause excessive heating of the Intel® Edison compute module.

The Intel® Edison compute module is a low power device. It normally operates at 200 mA. During Wi-Fi transmit bursts, the current could reach 600 mA for milliseconds. The sum of the Intel® Edison kit for Arduino* current, recharging, SD card, and shield power could exceed the 500 mA specification. This could cause triggering of the USB power switch within a PC, causing loss of USB functionality until the PC is restarted.

Some considerations of the power distribution in the Intel® Edison kit for Arduino*:

- There is a diode ORing of the 5 V DC/DC and the VBUS input. In the case of powering the Intel® Edison kit for Arduino* from VBUS, the shield voltage may be as low as 4.4 V (4.75 V VBUS min – 0.3 V diode drop). In the case of external power adapter, voltage to the shield will be 5 V ±2%.
- Using the Intel® Edison compute module as a USB HOST requires use of an external adapter.
- End-users are responsible for choosing a suitable battery and following all safety precautions, to prevent overcharging or charging when the battery temperature is too high. The battery should be at least 200 mAH capacity due to the 100 mA charging current. We recommend battery packs with internal protection circuits.

4.1 Boot voltage selection – DCIN signal

DCIN is a signal that indicates whether the Intel® Edison compute module is being powered from a battery or from an external power source. DCIN also sets the voltage level required on VSYS in order to boot. When DCIN is floating or tied to ground, the voltage on VSYS *must* rise from 2.5 to 3.5 V in 10 ms; otherwise the boot is aborted. When the boot is aborted, power must be cycled below 2.5 V. If DCIN is connected to VSYS, the Intel® Edison compute module will start to boot when VSYS is above 2.5 V for 100 ms.

Note: When DCIN is connected to VSYS, boot will occur whenever the voltage is above 2.8 V for 100 ms. The DCIN signal is attached to VSYS on the PCB.

Note: The absolute minimum voltage to assure Wi-Fi and Bluetooth functionality is 3.15 V.

§

5 Batteries

The rechargers chosen on the Intel® Edison kit for Arduino* and the Intel® Edison Breakout Board were designed for lithium-ion or lithium-polymer batteries. Follow the manufacturer's guidelines when charging batteries. Generally, charging current should not exceed 50 to 70% of the rated capacity. For example, a 200 mAH battery should be charged with 70% • 200 mA = (140 mA).

The Intel® Edison kit for Arduino* has a 100 mA charging current; the Intel® Edison Breakout Board has a 190 mA charging current.

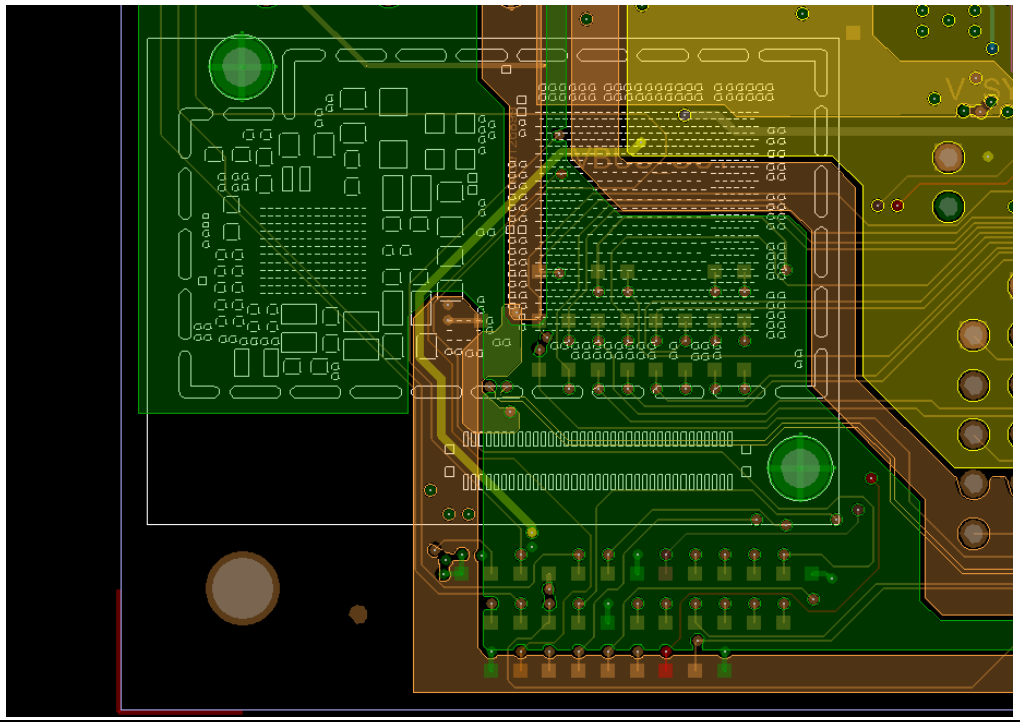
§

6 Layout

6.1 Antenna keepout

The area under and around the antenna should be kept free of all components, routes, and ground plane. The Intel® Edison compute module DXF in white with antenna keepout shown in the Arduino* trace layers. See Figure 7.

Figure 7 Area around antenna



6.2 Layout SD card, I2S, SPI, I2C

Table 10 Layout SD card

Signal parameter	Metric (mm)	Standard (mils)
Total length L1	0.254 to 101.6 mm	10 to 4000 mils
DATA/CMD/CTRL to CLK maximum pin-to-pin length mismatch	±2.54 mm	±100 mils
Minimum main route spacing ratio	60 × 60 µm. 1:1 trace width/space.	
CLK to DATA/CMD/CTRL matching	±200 mils	
Characteristic single ended impedance	42 to 45 ohm (±10%)	
Load capacitance	2 to 5 pF	

Note: 1) For SPI, total length is 6000 mils.
2) For I^2C, total length is 8000 mils.

6.3 LEDs

The Intel® Edison kit for Arduino has three LEDs. (See Figure 5 for locations.)

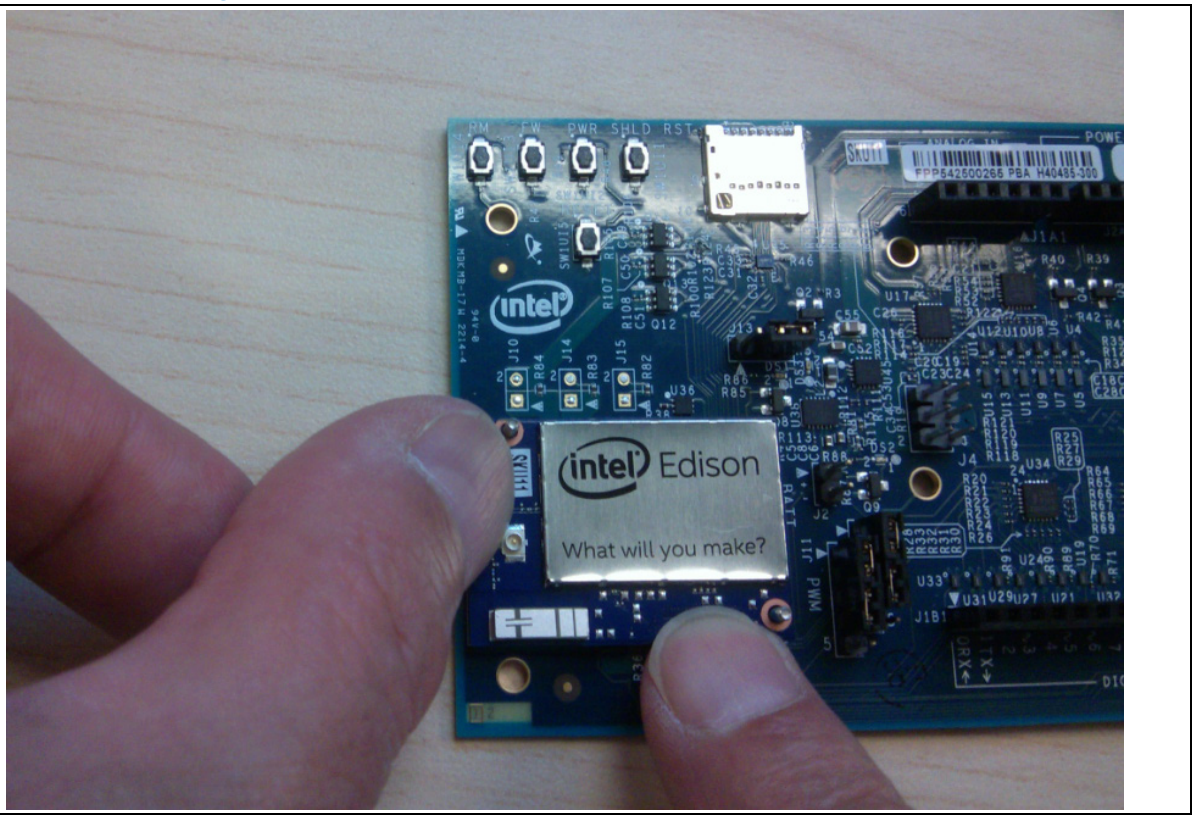
- DS1 is the reset LED. It will turn on when the Intel® Edison processor is running. When the processor is in reset and asserting RESET_OUT# low, it will turn off.
- DS2 is the standard LED on the Arduino* board. It runs using the 'blink' code or whenever Digital I/O 13 is asserted High. It can be used as an indicator under direct control.
- DS3 is the battery charging LED. It will turn on when the LTC4067 is charging an attached battery.

§

7 Handling

When assembling an Intel® Edison compute module to an Arduino* board, handle the Intel® Edison compute module by the PCB edges. Avoid holding or exerting pressure to the shields. To mate the Intel® Edison compute module to the Arduino* board, apply pressure directly above the connector and to the left corner, as shown in Figure 8.

Figure 8 Inserting an Intel® Edison compute module



§

8 Debug UART and Low-Power Sleep Mode

When the Intel® Edison compute module goes into low-power sleep, the UART internal FIFO and interface is powered down. Therefore, a two-wire UART (Rx/Tx) will lose the first received character whenever the Intel® Edison compute module is in low-power sleep mode. In order to avoid this condition, when sleep mode is enabled, a four-wire UART (Rx, Tx, CTS, and RTS) is required.

Note: Low-power sleep mode is disabled by default in the latest image. To address this, update your firmware as explained in the *Getting Started Guide* at *https://communities.intel.com/docs/DOC-23147*.

§

9 Buttons

This section explains the software functionality of the Intel® Edison kit for Arduino* buttons.

The Intel® Edison kit for Arduino* has the following buttons:

- **System reset.** Pressing the system reset button (SW1UI5) will reset the Intel® Edison compute module, and reset the I/O expanders, setting all the shield pins to high impedance state with no pullups.
- **Shield reset.** Pressing the shield reset button (SW1UI1) will pull the shield signal reset to the active low state. It does not affect the state of the Intel® Edison compute module or its I/O.
- **Power button.** The power button (SW1UI2) is configured by software. In general, pressing and holding this button will cause the Intel® Edison compute module to power down. (It will leave the I/O configuration in the port expanders in its current state.) Pressing this button momentarily when the Intel® Edison compute module is powered down (but power is still applied) will cause the Intel® Edison compute module to reboot. If the Intel® Edison compute module is running, then a momentary press will cause the Intel® Edison compute module to go into low power sleep mode. Pressing the button momentarily when the Intel® Edison compute module is asleep, will bring the Intel® Edison compute module into full power mode. You must press and hold SW1U15 for 8 seconds to reset the Intel® Edison compute module. Pressing the reset button for 4 seconds will restart the Intel® Edison compute module.

9.1 FWR_RCVR and RCVR_MODE

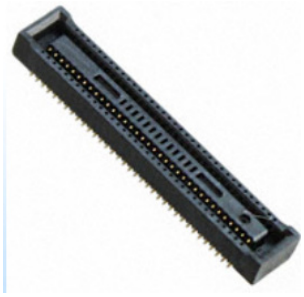
SW1UI3 and SW1UI4 are for factory use only.

§

10 Digikey sources

Figure 9 shows some third-party accessories you can use.

Figure 9 Digikey sources

 **Mating connector 2.0 mm** **DF40C(2.0)-70DS-0.4V(51) - H11908CT-ND Cut tape** **DF40C(2.0)-70DS-0.4V(51) H11908TR-ND Tape and Reelt**	 **Mini-breakout power jack PJ-002BH-SMT-TR** **CP-002BHPJCT-ND Cut tape** **PJ-002BH-SMT-TR CP-002BHPJTR-ND Tape and reel**
 Mini-breakout USB adapter cable **USB A female to Micro A male 10-00649 839-1105-ND**	 **Mini-breakout male header** **2x14 M20-9980745 952-1932-ND**

§

11 Shield pin configuration

This chapter will help you configure the Arduino* shield pins.

To configure the Arduino* shield pins, do the following:

1. Identify the Arduino* shield pin number of the pin you want to use, in the range IO0-IO19.
2. Identify the functions available for the given pin, and select the function you want to use. Typical functions are GPIO, PWM, UART, I^2C, SPI, ADC. Only some functions are available on each pin.
3. Determine which GPIO signals, if any, need to be configured to select the correct pin muxing option for the selected function. Some pins only have a single function, or do not require mux control.
4. Determine which GPIO signals, if any, need to be configured to select the pin buffer direction for input or output, and determine the direction that is required.
5. Determine which GPIO signals, if any, need to be configured to select the pullup resistor control, and whether the pullup resistor should be enabled or disabled. For most pin functions, the pullup resistors should typically be disabled. For GPIO input functions, the pullup resistor may optionally be enabled or disabled, according to your needs.
6. Export the above GPIO numbers for access in the Linux user-space environment (from the command shell).
7. Configure the above GPIO numbers for output.
8. Assert the TRI_STATE_ALL signal to disconnect the shield pins.
9. Set the above GPIO numbers to assert their output logic levels as high or low.
10. Set the SoC GPIO pin mode for the required functionality.
11. Deassert the TRI_STATE_ALL signal to reconnect the shield pins.

11.1 Configure IO5 as a GPIO input, with pullup resistor disabled

To configure IO5 as a GPIO input, with pullup resistor disabled, do the following:

1. Refer to Table 2 for the GPIO number. According to Table 2, the GPIO number for IO5 is 13.
2. According to Table 4, GPIO 43 pin-mux must be set to mode0 to select the GPIO.
3. According to Table 7, GPIO 253 must be set to 0 to disable the output direction for IO5.
4. According to Table 7, GPIO 221 must be set as a high-impedance input to disable the external pullup resistor for IO5.
5. According to Table 6, the TRI_STATE_ALL signal is controlled by GPIO 214.

After you have gathered all of this information, enter the following commands in Linux:

```
# echo 13 > /sys/class/gpio/export
# echo 253 > /sys/class/gpio/export
# echo 221 > /sys/class/gpio/export
# echo 214 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio214/direction
# echo low > /sys/class/gpio/gpio253/direction
# echo in > /sys/class/gpio/gpio221/direction
# echo mode0 > /sys/kernel/debug/gpio_debug/gpio13/current_pinmux
# echo in > /sys/class/gpio/gpio13/direction
# echo low > /sys/class/gpio/gpio214/direction
```

You should be able to use IO5 as a GPIO input. For example:

```
# cat /sys/class/gpio/gpio13/value
```

11.2 Configure IO11 as a GPIO input, with pullup resistor disabled

To configure IO11 as a GPIO input, with pullup resistor disabled, do the following:

1. Refer to Table 2 for the GPIO number. According to Table 2, the GPIO number for IO11 is 43.
2. According to Table 4, GPIO 262 must be set to 1 to select GPIO/SPI, GPIO 241 must be set to 0 to select GPIO, and GPIO 43 pin-mux must be set to 'mode0' to select GPIO.
3. According to Table 7, GPIO 259 must be set to 0 to disable the output direction for IO11.
4. According to Table 4, GPIO 227 must be set as a high-impedance input to disable the external pullup resistor for IO5.
5. According to Table 6, the TRI_STATE_ALL signal is controlled by GPIO 214.

After you have gathered all of this information, enter the following commands in Linux:

```
# echo 43 > /sys/class/gpio/export
# echo 262 > /sys/class/gpio/export
# echo 241 > /sys/class/gpio/export
# echo 259 > /sys/class/gpio/export
# echo 227 > /sys/class/gpio/export
# echo 214 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio214/direction
# echo high > /sys/class/gpio/gpio262/direction
# echo low > /sys/class/gpio/gpio241/direction
# echo mode0 > /sys/kernel/debug/gpio_debug/gpio43/current_pinmux
# echo low > /sys/class/gpio/gpio259/direction
# echo in > /sys/class/gpio/gpio227/direction
# echo in > /sys/class/gpio/gpio43/direction
# echo low > /sys/class/gpio/gpio214/direction
```

You should be able to use IO11 as a GPIO input. For example:

```
# cat /sys/class/gpio/gpio43/value
```

11.3 Configure IO7 as a GPIO input, with pullup resistor enabled

To configure IO7 as a GPIO input, with pullup resistor enabled, do the following:

1. Refer to Table 2 for the GPIO number. According to Table 2, the GPIO number for IO7 is 48.
2. According to Table 7, GPIO 255 must be set to 0 to disable the output direction for IO7.
3. According to Table 7, GPIO 223 must be set to output high to enable the external pullup resistor for IO7.
4. According to Table 6, the TRI_STATE_ALL signal is controlled by GPIO 214.

After you have gathered all of this information, enter the following commands in Linux:

```
# echo 48 > /sys/class/gpio/export
# echo 255 > /sys/class/gpio/export
# echo 223 > /sys/class/gpio/export
# echo 214 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio214/direction
# echo low > /sys/class/gpio/gpio255/direction
# echo high > /sys/class/gpio/gpio223/direction
# echo in > /sys/class/gpio/gpio48/direction
# echo low > /sys/class/gpio/gpio214/direction
```

You should be able to use IO7 as a GPIO input. For example:

```
# cat /sys/class/gpio/gpio48/value
```

11.4 Configure IO6 as a PWM output

To configure IO6 as a PWM output, do the following:

1. Refer to Table 2 for the GPIO number. According to Table 2, the GPIO number for IO6 is 182.
2. According to Table 4, GPIO 182 pin-mux must be set to 'mode1' to select PWM.
3. According to Table 7, GPIO 254 must be set to 1 to enable the output direction for IO6.
4. According to Table 7, GPIO 222 must be set as a high-impedance input to disable the pullup resistor for IO6.
5. According to Table 6, the TRI_STATE_ALL signal is controlled by GPIO 214.

After you have gathered all of this information, enter the following commands in Linux:

```
# echo 254 > /sys/class/gpio/export
# echo 222 > /sys/class/gpio/export
# echo 214 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio214/direction
# echo high > /sys/class/gpio/gpio254/direction
# echo in > /sys/class/gpio/gpio222/direction
# echo mode1 > /sys/kernel/debug/gpio_debug/gpio182/current_pinmux
# echo low > /sys/class/gpio/gpio214/direction
```

You should be able to use IO6 as a PWM output. For example:

```
# echo 2 > /sys/class/pwm/pwmchip0/export
# echo 2000000 > /sys/class/pwm/pwmchip0/pwm2/duty_cycle
# echo 1 > /sys/class/pwm/pwmchip0/pwm2/enable
```

11.5 Configure IO14 as an ADC input

To configure IO14 as an ADC input, do the following:

1. Refer to Table 2 for the GPIO number. According to Table 2, the GPIO number for IO14 is 44.
2. According to Table 4, GPIO 200 must be set to 1 to select ADC.
3. According to Table 7, GPIO 232 must be set to 0 to disable the output direction for IO14.
4. Any GPIO lines directly connected to IO14 should be configured as high-impedance inputs to prevent possible current leakage. According to Table 7, GPIO 208 is used to enable a pullup resistor for IO14.
5. According to Table 6, the TRI_STATE_ALL signal is controlled by GPIO 214.

After you have gathered all of this information, enter the following commands in Linux:

```
# echo 200 > /sys/class/gpio/export
# echo 232 > /sys/class/gpio/export
# echo 208 > /sys/class/gpio/export
# echo 214 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio214/direction
# echo high > /sys/class/gpio/gpio200/direction
# echo low > /sys/class/gpio/gpio232/direction
# echo in > /sys/class/gpio/gpio208/direction
# echo low > /sys/class/gpio/gpio214/direction
```

You should be able to use IO14 as an ADC input. For example:

```
# cat /sys/bus/iio/devices/iio:device1/in_voltage0_raw
```

Note: The default state of the mux switches and level-shifters for shield pins IO11 to 13 is inconsistent, and will impair SPI communication to the ADC if not configured properly. Thus, we recommend following the instructions elsewhere in this document to fully configure these pins for any of their functions (for example, SPI or GPIO) before attempting to use the ADC.

11.6 Configure IO18/IO19 for I²C connectivity

To configure IO18 and IO19 for I²C connectivity, do the following:

1. Refer to Table 2 for the GPIO numbers. According to Table 2, the GPIO numbers for IO18 and IO19 are 28 and 27, respectively.
2. According to Table 4, GPIO 204 must be set to 1 to select GPIO/I²C, and GPIO 28 pin-mux must be set to 'mode1' to select I²C for IO18.
3. According to Table 4, GPIO 205 must be set to 1 to select GPIO/I2C, and GPIO 27 pin-mux must be set to 'mode1' to select I²C for IO19.
4. GPIO 14 and GPIO 165 are also connected to the I²C signals, and should be configured as high-impedance inputs when I²C is in use on these pins, to prevent them driving a signal on the I²C bus.
5. According to Table 7, GPIO 236 must be set to 0 to disable the output direction for GPIO 14, and GPIO 237 must be set to 0 to disable the output direction for GPIO 165.
6. According to Table 7, GPIO 212 and 213 must be set as high-impedance inputs to disable the pullup resistors for IO18 and IO19, respectively.
7. According to Table 6, the TRI_STATE_ALL signal is controlled by GPIO 214.

After you have gathered all of this information, enter the following commands in Linux:

```
# echo 28 > /sys/class/gpio/export
# echo 27 > /sys/class/gpio/export
# echo 204 > /sys/class/gpio/export
# echo 205 > /sys/class/gpio/export
# echo 236 > /sys/class/gpio/export
# echo 237 > /sys/class/gpio/export
# echo 14 > /sys/class/gpio/export
# echo 165 > /sys/class/gpio/export
# echo 212 > /sys/class/gpio/export
# echo 213 > /sys/class/gpio/export
# echo 214 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio214/direction
# echo high > /sys/class/gpio/gpio204/direction
# echo high > /sys/class/gpio/gpio205/direction
# echo in > /sys/class/gpio/gpio14/direction
# echo in > /sys/class/gpio/gpio165/direction
# echo low > /sys/class/gpio/gpio236/direction
# echo low > /sys/class/gpio/gpio237/direction
# echo in > /sys/class/gpio/gpio212/direction
# echo in > /sys/class/gpio/gpio213/direction
# echo mode1 > /sys/kernel/debug/gpio_debug/gpio28/current_pinmux
# echo mode1 > /sys/kernel/debug/gpio_debug/gpio27/current_pinmux
# echo low > /sys/class/gpio/gpio214/direction
```

You should be able to use IO18 and IO19 for I²C communication.

11.7 Configure IO10 through IO13 for SPI connectivity

To configure IO10 through IO13 for SPI connectivity, do the following:

1. Refer to Table 2 for the GPIO numbers. According to Table 2, the GPIO numbers for IO10 through IO13 are 111, 115, 114, and 109, respectively.
2. According to Table 4, GPIO 263 must be set to 1 to select GPIO/SPI, GPIO 240 must be set to 1 to select SPI, and GPIO 111 pin-mux must be set to 'mode1' to select SPI for IO10.
3. According to Table 4, GPIO 262 must be set to 1 to select GPIO/SPI, GPIO 241 must be set to 1 to select SPI, and GPIO 115 pin-mux must be set to 'mode1' to select SPI for IO11.
4. According to Table 4, GPIO 242 must be set to 1 to select SPI, and GPIO 114 pin-mux must be set to 'mode1' to select SPI for IO12.
5. According to Table 4, GPIO 243 must be set to 1 to select SPI, and GPIO 109 pin-mux must be set to 'mode1' to select SPI for IO13.
6. According to Table 7, GPIO 258 must be set to 1 to enable the output direction for IO10, GPIO 259 must be set to 1 to enable the output direction for IO11, GPIO 260 must be set to 0 to disable the output direction for IO12, and GPIO 261 must be set to 1 to enable the output direction for IO13.
7. According to Table 7, GPIOs 226 through 229 must be set as high-impedance inputs to disable the pullup resistors for IO10 through IO13.
8. According to Table 6, the TRI_STATE_ALL signal is controlled by GPIO 214.
9. After you have gathered all of this information, enter the following commands in Linux:

```
# echo 111 > /sys/class/gpio/export
# echo 115 > /sys/class/gpio/export
# echo 114 > /sys/class/gpio/export
# echo 109 > /sys/class/gpio/export
# echo 263 > /sys/class/gpio/export
# echo 240 > /sys/class/gpio/export
# echo 262 > /sys/class/gpio/export
# echo 241 > /sys/class/gpio/export
# echo 242 > /sys/class/gpio/export
# echo 243 > /sys/class/gpio/export
# echo 258 > /sys/class/gpio/export
# echo 259 > /sys/class/gpio/export
# echo 260 > /sys/class/gpio/export
# echo 261 > /sys/class/gpio/export
# echo 226 > /sys/class/gpio/export
# echo 227 > /sys/class/gpio/export
# echo 228 > /sys/class/gpio/export
# echo 229 > /sys/class/gpio/export
# echo 214 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio214/direction
# echo high > /sys/class/gpio/gpio263/direction
# echo high > /sys/class/gpio/gpio240/direction
# echo high > /sys/class/gpio/gpio262/direction
# echo high > /sys/class/gpio/gpio241/direction
# echo high > /sys/class/gpio/gpio242/direction
# echo high > /sys/class/gpio/gpio243/direction
# echo high > /sys/class/gpio/gpio258/direction
# echo high > /sys/class/gpio/gpio259/direction
# echo low > /sys/class/gpio/gpio260/direction
# echo high > /sys/class/gpio/gpio261/direction
# echo in > /sys/class/gpio/gpio226/direction
# echo in > /sys/class/gpio/gpio227/direction
```

```
# echo in > /sys/class/gpio/gpio228/direction
# echo in > /sys/class/gpio/gpio229/direction
# echo mode1 > /sys/kernel/debug/gpio_debug/gpio111/current_pinmux
# echo mode1 > /sys/kernel/debug/gpio_debug/gpio115/current_pinmux
# echo mode1 > /sys/kernel/debug/gpio_debug/gpio114/current_pinmux
# echo mode1 > /sys/kernel/debug/gpio_debug/gpio109/current_pinmux
# echo low > /sys/class/gpio/gpio214/direction
```

You should be able to use IO10 through IO13 for SPI connectivity.

§